The Theory of Twiners and Linear Parametricity (Note) (Towards new interaction between category theory and proof theory)

AUTHOR(S):

Hasegawa, Ryu

CITATION:

Hasegawa, Ryu. The Theory of Twiners and Linear Parametricity (Note) (Towards new interaction between category theory and proof theory). 数理解析研究所講究録 2001, 1217: 23-36

ISSUE DATE:

2001-06

URL:

http://hdl.handle.net/2433/41229

数理解析研究所講究録 1217 巻 2001 年 23-36

The Theory of Twiners and Linear Parametricity (Note)

Ryu Hasegawa (長谷川 立)

Graduate School of Mathematical Sciences, The University of Tokyo,
(東京大学 数理科学研究科),
Komaba 3-8-1, Meguro-ku, Tokyo 153-8914, Japan

1 Introduction

We want to show that solutions of domain equations, or equivalently existence of recursive types, can be derived from a computer theoretic concept, called the principle of *linear parametricity.* Existence of recursive types has applications in, for example, semantics of functional programming, type-theoretic study of object-oriented programming, and the theory of programme transformation, as reviewed in the following. In usual, the existence of recursive types is ensured using a mathematical theory of Scott domains or alike. Our innovation lies in that it can be derived from a more computer theoretic concept. In this work, we focus on giving a model satisfying the principle of linear parametricity in a reasonable sense, leaving more syntax-oriented matters such as axiomatization to a future work.

In many activities of computing, we often need the structures defined recursively. Simple data structures such as lists, trees, and even natural numbers are examples of recursively defined data. These examples are rather easily rationalized, since they are given as fixed points of operators where indeterminates occur only positively. A naive set-theoretic understanding is sufficient for them in most occasions.

A more challenging problem is whether we can give rationale for fixed points of operators that may contain negative occurrences as well. One of early speculations leading to this problem was to provide semantics for the untyped lambda calculus. The calculus has a rather peculiar structure where functions and data have no distinction and everything can be applied to everything. This observation leads us to the fact that giving semantics of the untyped lambda calculus amounts to detecting an entity X that is isomorphic to

the function space $X \Rightarrow X$ on itself. Namely we need a fixed point of the operator $F(X)$ defined by $X \Rightarrow X$ where the indeterminate X occurs both positively and negatively. If we work in set theory and interpret $X \Rightarrow X$ as the set of all functions, simple cardinality argument reveals that there are only trivial solutions where the fixed point X is a singleton. Hence we must give away naive set-theoretic interpretation, and try to find certain exotic structures in which we can have fixed points of the operators of the type above. It was late sixties that Scott finally gave a construction by which we can form fixed-points of such operators [18]. His construction uses certain (not even Hausdorff) topological spaces and continuous functions. Later the idea leaded to evoluation of domain theory [7], which nowadays is matured so as to be used as fundamental tools to give mathematical reasoning to programming languages.

Turning to more up-to-date topics, we still hold the same sort of problems in several situations. One of them comes from a type-theoretic study of object-oriented programming. During the attempt to pin down the essence of the object-oriented programming in functional programming setting, several new ideas are discovered and imported into the traditional theory. These include how to incorporate the concepts of inheritance and self-reference. At an early stage, unfortunately, in the author's opinion, inheritance was emphasized too much. It was a vogue to adjoin the structure of partial orders to systems in order to interpret inheritance. However the author believes the concept of self-reference is equally or more important as the heart of object-oriented programming and should be dealt with in suitable respects (to be fair, we remark that many systems pay attention to also self-reference properly). A philosophy of object-orientedness is that each definition of object should be self-contained (putting inheritance aside) so that the behaviour of an object is determined by information written in the definition of the object itself. As a natural consequence, there should be a mechanism with which certain methods, often called binary methods, can refer the object itself. For example, the definition of the objects of queues should contain the method to update queues, this method accessing the object itself and modifying it. To comprehend this phenomenon type-theoretically, we usually employ recursive types where indeterminates may occur negatively [1, 15, 17]. Of course, the consistency of recursive types is a sensitive matter. A conceivable way to check it is to build mathematical semantics, employing domain theory.

Another example is taken from the theory of programme transformation. The fusion rule is a machinery to eliminate intermediate structures passed between two functional procedures, when they are composed. The rule asserts that, if two canonical functions from inductive types satisfy certain commutativity conditions, we can combine them into a single function so that intermediate return values are never created. Under lazy evaluation of functional programming, we can use also coinductive types as potentially infinite data types. The corresponding fusion rules can be defined for coinductive types as well. What is more interesting, however, occurs when inductive and coinductive types interact. Hylomorphism is by definition the composition of a canonical function into coinductive types $\nu X.\, F(X)$ followed by a canonical function from inductive types $\mu X.\, F(X)$, provided that these coinductive and inductive types coincide, that is $\nu X.\, F(X) = \mu X.\, F(X)$ [4, 11]. An advantage of hylomorphism is that intermediate structures can be always eliminated. But, to use this notion, we must assume that inductive types coincide coinductive types. At first sight, this assumption looks rather unnatural. For example, intuitively, the inductive type for the operator $1 + A \times X$ is that of finite lists of members of A while the coinductive type is that of possibly infinite sequences. We can show that the assumption of coincidence is equivalent to existence of recursive types under certain conditions. To ensure the notion of hylomorphism to be consistent, we must appeal to domain theory again.

Therefore many subjects rely on exquisite theory of Scott domains; from old problem of giving semantics to the untyped lambda calculus, to newer fields of theoretical computer science, e.g., object-oriented programming and the theory of programme transformation. There is no doubt that domain theory was one of the most successful theories to give rigid foundations to the theory of programming. At this point, however, we want to address a fundamental query. Can the theory of programming be built only on Scott domains? Trained theoreticians may fully use domain theory as a vehicle for verification, whereas the working programmers (even the theoreticians themselves when they write actual codes) would not take Scott domains into account to understand how the programmes they write work. A role of semantics is to give an intuition of the behaviour of programmes. In this respect, domain theory is too apart from computational concepts programmers naturally bear in mind.

Our goal is to propose a computer theoretic concept from which we can derive the construction usually achieved by domain theory. Namely we want to have the same effect as domain thoery without appealing to Scott domains. The new concept is called linear parametricity. An advantage of our method is that we can obtain various results from a single principle, and that the principle of linear parametricity can be understood from a computer theoretic point of view.

Earlier we investigated the notion of (full) parametricity for polymorphic programming languages. The intuition behind parametricity can be easily understood computationally. That is, a polymorphic programme is called parametric if it makes no explicit use of information of types. Then the principle of parametricity assures that, if a polymorphic programme is parametric, it behaves in a uniform way, irrelevant of the types with which we substitute type-parameters. In earlier works, we studied the principle of parametricity from several perspectives, including categorical and logical ones, and demonstrated that this simple principle induce many nice properties [8, 9, 10]. However the principle of parametricity can coexist solely with a fragment of polymorphic languages where only terminating programmes matter. The property of languages allowing recursive programming contradicts to the categorical consequences of the principle.

Linear parametricity is a reduced version of full parametricity, and does not contradict to existence of recursive programmes. Furthermore, in accordance with recursive programmes, linear parametricity yields better results: solutions of domain equations, i.e, recursive types, which cannot be consequences of full parametricity. Linearity means the same thing as that in linear logic [5]. Namely linear parametricity is the principle of parametricity asserted in the context of linear logic. A similar approach is proposed in [16].

In this work, we develop a model of second order linear logic, using new mathematical stuffs called *twiners*, which may be interesting in their own right. They are extensions of Joyal's analytic functors [12, 13] and Girard's normal functors [6]. Into the twiner model of second order linear logic, we incorporate the notion of linear parametricity, and form a second more elaborate model satisfying the principle of linear parametricity. Finally we verify that, in the new model, we can solve domain equations using syntax of linear logic, rather than Scott D_∞-style construction. It would take dozens of pages

to fully expand the theory of twiners. So, in this paper, we must be content with overview of the theories, leaving details to the full paper in prepration.

2 First Model

We assume certain amount of knowledge in 2-category and bicategory theory. We refer the reader to standard literatures [14, 3]. To fix terminology, we use the following terms: pseudo-functors, quasi-natural transformations, and modifications.

A *groupoid* is a small category where every morphism has an inverse. In particular, if a groupoid has a single object only, the set of all morphisms forms a group. Conversely every group may be regarded as a groupoid with a single object. We note that, since every morphism of a groupoid A is invertible, the opposite category A^{op} is equivalent to A as groupoids. To distinguish objects etc. in A^{op} from those in A, we often use notation $\bar{x}$ for the thing in A^{op} corresponding to x in A. We sometimes write the overline as in $\overline{A}$ to denote the opposite groupoid itself.

A *groupoid-enriched* category is such that its homsets are endowed with the structure of groupoids satisfying certain conditions we do not specify here. It comes from standard enriched category theory by taking the category of all groupoids and all groupoid homomorphisms with cartesian product as a monoidal structure. We note that a groupoid-enriched category is nothing else than a 2-category where every 2-cells are invertible.

Example: (i) **Gpoid** is the groupoid-enriched category of all groupoids, all groupoid homomorphisms, and all natural transformations which turns out automotically to be isomorphisms.

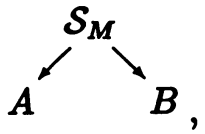
(ii) For each groupoid A, the groupoid-enriched category $\mathbf{Gpoid}^A$ is given by all pseudo-functors on A into **Gpoid**, all quasi-natural transformations, and all modifications. By an analogy that an object of $\mathbf{Set}^G$ for a group G is the same thing as a G-set, we call an object of $\mathbf{Gpoid}^A$ an A-groupoid.

(iii) We can define the slice groupoid-enriched category $\mathbf{Gpoid}/A$, the objects of which are groupoid homomorphisms $T \xrightarrow{f} A$ on some groupoid T. We omit the details here.

2.1 Proposition
Let A be a groupoid.

Biequivalence $\mathbf{Gpoid}^A \cong \mathbf{Gpoid}/A$ *between groupoid-enriched categories holds.*

The right-to-left direction of the biequivalence is given by the Grothendieck construction [2]. For each object t in $\mathbf{Gpoid}^A$, we denote the groupoid over A obtained as the Grothendieck construction by either $\mathrm{Gr}(t)$ or, bollowing integral notation, $\int_{x \in A} t[x]$. In particular, an $A^{\mathrm{op}} \times B$-groupoid M corresponds to a span



taking $A \cong A^{\mathrm{op}}$ into considertation. We call $A^{\mathrm{op}} \times B$-groupoids M *biprofunctors* from A into B as a generalization of profunctors, and write $M : A \multimap B$. We might regard such a biprofunctor M as a matrix of A columns and B rows. If N is a biprofunctor from B into C, we define the matrix composition $NM : A \multimap C$ by $(NM)[\bar{x}, z] := \int_{y \in B} N[\bar{y}, z]M[\bar{x}, y]$. In terms of spans, the matrix composition corresponds to taking a bipullback $\mathcal{S}_M \times_B \mathcal{S}_N$.

We can give the interpretation of the additive-multiplicative fragment of linear logic at this point. Each type is interpreted as a groupoid A. A term of type A is interpreted as an A-groupoid, that is, an object of $\mathbf{Gpoid}^A$. We give the interpretations of types only, leaving those of terms to the full paper. If we identify types A with their interpretations, the interpretations are given as follows:

$$
\begin{aligned}
[\![A^{\perp}]\!] &= A^{\mathrm{op}} \\
[\![A \otimes B]\!] &= [\![A \parr B]\!] = A \times B \\
[\![A \,\&\, B]\!] &= [\![A \oplus B]\!] = A + B \\
[\![1]\!] &= [\![\bot]\!] = 1 \\
[\![\top]\!] &= [\![0]\!] = \emptyset.
\end{aligned}
$$

In particular, the interpretation of linear implication is given as $[\![A \multimap B]\!] = A^{\mathrm{op}} \times B$. Namely a term of type $A \multimap B$ is interpreted as a biprofunctor from A into B.

To define the intepretation of exponential in linear logic, we introduce wreath product of groupoids. Let G be a groupoid endowed with a pseudo-functor $G \xrightarrow{\varphi} \mathbf{Gpoid}$. The *wreath product* A wr G is defined by the Grothendieck construction $\int_{x \in G} \mathbf{Gpoid}(\varphi(x), A)$. We remark that this concept is a generalization of the traditional wreath product of groups, occurring as a special case of semidirect product [19]. To be more general, if $\mathbf{C}$ is a groupoid-enriched category, we can define wreath product A wr G for each object $A \in \mathbf{C}$ and a pseudo-functor $G \xrightarrow{\varphi} \mathbf{C}$ from a groupoid G, simply by putting the groupoid A wr G to be $\int_{x \in G} \mathbf{C}(\varphi(x), A)$. For the interpretation of linear logic, we must slightly extend groupoid G in definition of wreath product. We allow G to have a family C of empty components. Namely we formally consider G to be a pair of a groupoid G' and a set C. The contribution of one empty component to wreath product is a singleton as a sum over an empty set. Hence, for such a pair G, we define A wr G to be a direct sum of A wr G' and a disjoint groupoid (i.e., a set) C.

2.2 Definition (of S)
An extended groupoid S is a direct sum of symmetric groups S_n where n ranges over the set $\mathbb{N}$ of natural numbers. For $n = 0$, we regard S_0 to be an empty component in the sense above.

The interpretation of exponential is given as wreath product $[\![!A]\!] = A$ wr S. Accordingly $[\![?A]\!] = A$ wr S^{op}, which is equivalent to A wr S.

Next we turn to second order variable types. We want to consider types such as $!X$ where X is a type variable. To interpret $!X$, we are interested in the operation $X \mapsto (X \,\mathrm{wr}\, S)$. It turns out that $(\text{-})\,\mathrm{wr}\, S$ is a 2-functor from **Gpoid** to itself. In general, (-) wr G for $G \xrightarrow{\varphi} \mathbf{C}$ is a 2-functor on $\mathbf{C}$ into **Gpoid**

In a groupoid-enriched category $\mathbf{C}$, we call a 1-cell $A \xrightarrow{f} B$ *essentially onto* if the groupoid homomorphism $\mathbf{C}(B, X) \to \mathbf{C}(A, X)$ induced by composition is faithful for every object X. Likewise we call f *surjective* if $\mathbf{C}(B, X) \to \mathbf{C}(A, X)$ is full and faithful for every X. An object A is *biprojective* iff the 2-functor $\mathbf{C}(A, \text{-})$ carries surjections to surjections. We call A *quasi-biprojective* iff $\mathbf{C}(A, \text{-})$ carries surjections to essentially onto groupoid homomorphisms. Moreover we call A *finitely bipresentable* iff $\mathbf{C}(A, \text{-})$ preserves filtered bicolimits.

2.3 Theorem
Let $\mathbf{C}$ *be a groupoid-enriched category having all bilimits and all filtered bicolimits, and let* $\mathbf{C} \xrightarrow{F} \mathbf{Gpoid}$ *be a pseudo-functor.*

The following are equivalent.

(i) *The pseudo-functor* F *is quasi-naturally equivalent to the 2-functor* (-)wr G *associated to a pseudo-functor* $G \xrightarrow{\varphi} \mathbf{C}$ *subject to the condition that* $\varphi(x)$ *is finitely bipresentable for every object* $x \in G$.

(ii) *The pseudo-functor* F *preserves filtered bicolimits and bipullbacks.*

(iii) *For each object* (A, x) *of the Grothendieck construction* $\mathrm{Gr}(F)$, *the slice groupoid-enriched category* $\mathrm{Gr}(F)/(A, x)$ *has a biinitial object* $(Z, c) \xrightarrow{(k,\alpha)} (A, x)$ *where* Z *is finitely bipresentable.*

2.4 Definition
Let $\mathbf{C}$ be a groupoid-enriched category having all bilimits and all filtered bicolimits.

A *twiner* on $\mathbf{C}$ is a pseudo-functor $\mathbf{C} \xrightarrow{F} \mathbf{Gpoid}$ satisfying one of the equivalent conditions in the preceding theorem. A twiner is called *discrete* iff it carries surjections to surjections. A twiner is called *quasi-discrete* iff it carries surjections to essentially onto groupoid homomorphisms.

A twiner (-) wr G for $G \xrightarrow{\varphi} \mathbf{C}$ is discrete iff all $\varphi(x)$ are biprojective. It is quasi-discrete iff all $\varphi(x)$ are quasi-biprojective.

Now we give the interpretation of types and terms having type variables in linear logic. A type $F(X_1, X_2, \ldots, X_n)$ with n type variables is interpreted as a discrete twiner on $\mathbf{Gpoid}^n$. Moreover a term of type $F(X_1, X_2, \ldots, X_n)$ is interpreted as a quasi-discrete twiner on the Grothendieck construction $\mathrm{Gr}(F)$. To $\mathrm{Gr}(F) \xrightarrow{t} \mathbf{Gpoid}$ and a groupoid A, we can associate a FA-groupoid t_A by definition $t_A[x] := t(A, x)$.

For a later use, we note that a twiner extends to operations on biprofunctors. If F is a twiner on **Gpoid** and $M : A \multimap B$ is a biprofunctor, we can associate $F_{bipro}M : FA \multimap FB$. In fact, M corresponds to a span $\mathcal{S}_M$ over A and B. So we simply define $F_{bipro}M$ to correspond to $F(\mathcal{S}_M)$ over FA and FB. In the sequel, we write simply FM instead of cumbersome $F_{bipro}M$.

Finally we must give the interpretation of second order quantified types. Indeed we give two interpretations. The first one stated here is based on the observations summarized so far. Later we provide with more elaborate construction so that the model enjoys the principle of linear parametricity. Let $B_{qb}(\mathbf{Gpoid})$ be the 2-groupoid of all quasi-biprojective, finitely bipresentable groupoids, all equivalences between such groupoids, and all natural isomorphisms. We note that every quasi-biprojective groupoid is equivalent to a direct sum of free groups. The 2-groupoid $B_{qb}(\mathbf{Gpoid})$ is actually biequivalent to a groupoid, as a consequence of the fact that all free groups have trivial centers. For each discrete twiner F on **Gpoid**, we define groupoid πF by

$$\pi F \;=\; \textstyle\int_{X \in B_{qb}(\mathbf{Gpoid})} F(X).$$

Then the interpretation of second order quantified types is given as $[\![\forall X.\, F(X)]\!] = \pi F$. Now we have obtained the first model of second order linear logic.

2.5 Theorem
The interpretation above gives a sound model of second order linear logic.

3 Linear Parametricity and Recursive Types

We truncate the first model to the one satisfying linear parametricity. First we introduce a linear dependent type theory with realizability semantics informally. Types are identified with groupoids and terms of type A are objects of groupoid-enriched category $\mathbf{Gpoid}^A$. The only logical connective we are concerned with is the linear first order universal quantification $(\forall x : A)\varphi(x)$. As an atomic formula, we take equality predicate $t =_A u$ for each type A. We interpret each formula as a groupoid. For equality, the formula $t =_A u$ is interpreted as the groupoid

$$\langle u|t\rangle := \textstyle\int_{x \in A} \overline{u[x]}t[x]$$

where the concatenation is an abbreviation of direct product of two groupoids. We note that there is an embedding $A^{\mathrm{op}} \to \mathbf{Gpoid}^A$ carrying $\bar{x}$ to the A-groupoid denoted by $\{\bar{x}\}$ which carries $a \in A$ to the discrete groupoid $A(x, a)$. For interpretation of linear quantifier, if $\varphi(x)$ is interpreted as

F : $\mathbf{Gpoid}^A \to \mathbf{Gpoid}$, then the formula $(\forall x\!:\!A)\varphi(x)$ is interpreted by the Grothendieck construction $\int_{x\in A} F\{\bar{x}\}$. In particular, equality $t =_\perp u$ for type $\perp$ (that is, a singleton 1) is interpreted by $u^{\mathrm{op}} \times t$. Then $t =_A u$ may be regarded as an acronym of $(\forall x\!:\!A^\perp)(tx =_\perp ux)$.

So far linearity does not come on the scene yet. It involves the witness relation $s \Vdash \varphi$ we introduce below, read as s witnesses φ. Here s is an A-groupoid if the formula φ is interpreted by groupoid A. For equality predicate, $s \Vdash (t =_\perp u)$ iff s is an identity biprofunctor from u into t. For the linear quantifier, the witness relation $s \Vdash (\forall x\!:\!A)\varphi(x)$ holds for an $(\int_{x\in A} F\{\bar{x}\})$-groupoid s iff $su \Vdash Fu$ holds for every $u \in \mathbf{Gpoid}^A$. Here an Fu-groupoid su is defined as

$$su[y] \;=\; \iint_{x\in A,\, c\in F\{\bar{x}\}} s[x,c] \int_{k\in u[x]} Fu(Fk(c),y)$$

for y in the groupoid Fu. We note $u[x]$ equals $\mathbf{Gpoid}^A(\{\bar{x}\},u)$. So Fk is a groupoid homomorphism from $F\{\bar{x}\}$ into Fu. We emphasize that linearlity involves witnesses only. There may be several occurrences of x in the linearly quantified formula $(\forall x\!:\!A)\varphi(x)$, including the case of null occurrence. The null case, say $(\forall x\!:\!A)B$, is written linear implication $A \multimap B$.

A (binary) *linear predicate* between types A and B is a predicate $P(x,y)$ for $x\!:\!A$ and $y\!:\!B$, endowed with its interpretation as a biprofunctor $M : A \multimap\!\!\!\!\to B$ and witness relation $s \Vdash P(t,u)$ for each pair of t and t'. Here s is a $\langle u|M|t\rangle$-groupoid, where $\langle u|M|t\rangle$ is defined to be $\iint_{x\in A, y\in B} \overline{u[y]}M[\bar{x},y]t[x]$.

Example: (i) $x =_A y$ is a linear predicate between A and A. It is interpreted as an identity biprofunctor, and witness relation has been defined above.

(ii) If R and S are linear predicates interpreted by biprofunctors $M : A \multimap\!\!\!\!\to A'$ and $N : B \multimap\!\!\!\!\to B'$, then $R \otimes_0 S$ is a binary predicate between $A \otimes B$ and $A' \otimes B'$. Its interpretation is $M \otimes N$ defined as $(M \otimes N)[\bar{x},\bar{y},x',y'] := M[\bar{x},x']N[y,y']$. Witness relation is defined to hold for $r \otimes s \Vdash R(t,t') \otimes S(u,u')$ iff both $r \Vdash R(t,t')$ and $s \Vdash S(u,u')$ hold.

(iii) If R is a linear predicate between A and A', the *dual* $R^\perp$ between A^{op} and A'^{op} is defined. We regard

$$R^\perp(t,t') \;=\; (\forall x\!:\!A)(\forall x'\!:\!A')\ (R(x,x') \multimap (tx =_\perp t'x')),$$

from which its interpretation and witness relation are induced.

3.1 Definition
A *factual* predicate is a linear predicate R satisfying that $s \Vdash R(t,t')$ iff $s \Vdash R^{\perp\perp}(t,t')$.

As a basic observation, the double dual $R^{\perp\perp}$ is a factual predicate for every linear predicate R. Using this observation, we associate factual predicates $F(R)$ to each type $F(X)$ with a type variable X and each factual predicate R. We have given a linear predicate $R \otimes_0 S$ in the examples above. Likewise we can define linear predicates $R \,\&_0\, S$ and $!_0 R$ in a straightforward way. Definition of $\forall_0 Y.\, F(R,Y)$ is more complicated, and we omit it here. These predicates with suffix 0 are not factual. So we define as follows:

$$\begin{aligned}
F(R) \parr G(R) &= (F(R)^\perp \otimes_0 G(R)^\perp)^\perp \\
F(R) \otimes G(R) &= (F(R)^\perp \parr G(R)^\perp)^\perp \\
\\
F(R) \oplus G(R) &= (F(R)^\perp \,\&_0\, G(R)^\perp)^\perp \\
F(R) \,\&\, G(R) &= (F(R)^\perp \oplus G(R)^\perp)^\perp \\
\\
?F(R) &= (!_0 F(R)^\perp)^\perp \\
!F(R) &= (?F(R)^\perp)^\perp \\
\\
\exists Y.\, G(R,Y) &= (\forall_0 Y.\, G(R,Y)^\perp)^\perp \\
\forall Y.\, G(R,Y) &= (\exists Y.\, G(R,Y)^\perp)^\perp
\end{aligned}$$

If a linear predicate R is interpreted by $M : A \multimap B$, then $F(R)$ is interpreted by the biprofunctor FM.

We extend a twiner t on $\mathrm{Gr}(F)$ to operations on biprofunctors, as a twiner F on **Gpoid** extends to $F_{bipro}M = FM$. For each biprofunctor $M : A \multimap B$, we want to define a $\langle t_B | FM | t_A \rangle$-groupoid t_M. In terms of spans, the groupoid $\langle t_B | FM | t_A \rangle$ corresponds to bipullback $T_A \times_{FA} F\mathcal{S}_M \times_{FB} T_B$ where by convention $t_A \in \mathbf{Gpoid}^{FA}$ corresponds to a groupoid T_A over FA and so on. Then t_M is defined to correspond $T_{\mathcal{S}_M}$ factoring through $T_A \times_{FA} F\mathcal{S}_M \times_{FB} T_B$.

3.2 Definition
Let t be a quasi-discrete twiner on $\mathrm{Gr}(F)$ where F is a discrete twiner on

t is *linearly parametric* iff witness relation $t_M \models F(R)(t_A, t_B)$ holds for every factual linear predicate R interpreted by $M : A \multimap B$.

We recall that a twiner is equivalent to a 2-functor given as wreath product. It has the shape (-) wr $G = \int_{z \in G} h^{\varphi(z)}$ where $h^{\varphi(z)}$ is the representable 2-functor $h^{\varphi(z)}(A, x) = \mathrm{Gr}(F)(\varphi(z), (A, x))$. The following is our main lemma towards the principle of linear parametricity.

3.3 Lemma
Let t be a quasi-discrete twiner on $\mathrm{Gr}(F)$, given as wreath product $\int_{z \in G} h^{\varphi(z)}$. Then t is linearly parametric iff a representable 2-functor $h^{\varphi(z)}$ is linear parametric for every object $z \in G$.

We define $\pi^P F$ be the full subgroupoid of πF of all objects (Z, c) satisfying that the representable 2-functor $\mathrm{Gr}(F)((Z, c), \text{-})$ is linearly parametric. With this lamme, a quasi-discrete twiner (-) wr G induced by $G \xrightarrow{\varphi} \mathrm{Gr}(F)$ is linearly parametric iff the image of φ is contained in $\pi^P F$. Now we define the second model of second order linear logic simply by changing the interpretation of quantifier. We interpret $\forall X. F(X)$ by $\pi^P F$. We can verify that the interpretation of every term is actually linearly parametric, that is:

3.4 Theorem
The second interpretation given above provides a sound model of second order linear logic.

An advantage of our linearly parametric model to earlier models satisfying full parametricity is that we can have fixed-point operator, which is needed to interpret recursive programmes.

3.5 Theorem
We have a linearly parametric fixed-point combinator fix *of type* $\forall X. !(!X \multimap X) \multimap X$.

We can represent recursive types in the linearly parametric second model as follows. Let $F(X)$ be a type where X may occur both positively and negatively. Separating positive occurrences X^+ and negative occurrences X^-, we may write $F(X^-, X^+)$. We define two types A and B as

$$\begin{aligned} A &= \exists X, Y.!(X \multimap F(Y,X)) \otimes !(F(X,Y) \multimap Y) \otimes X \\ B &= \forall X, Y.!(X \multimap F(Y,X)) \otimes !(F(X,Y) \multimap Y) \multimap Y. \end{aligned}$$

Then we can verify that $B \cong F(A,B)$ and $A \cong F(B,A)$. The proof simulates that of representation of initial algebras and final coalgebras under full parametricity. Moreover $A \cong B$ holds in the linearly parametric model, as a consequence of the existence of the fixed-point combinator fix. Thus the following holds:

3.6 Theorem

Suppose that groupoids A and B are given as above.

Then $A \cong F(A)$ holds in the linearly parametric model of twiners. Namely A gives an encoding of recursive type $\mathrm{rec}\, X.F(X)$ in the framework of second order linear logic augmented with a fixed-point combinator.

References

[1] M. Abadi and L. Cardelli, A theory of primitive objects, Second-order systems, *Sci. Computer Programming* **25** (1995) 81–116.

[2] M. Barr and C. Wells, *Category Theory for Computing Science*, Prentice-Hall International Series in Computer Science, (Prentice-Hall, 1990).

[3] J. Bénabou, Introduction to bicategories, in: *Reports of the Midwest Category Seminar*, (Springer, 1967) pp. 1–77.

[4] R. S. Bird, Functional algorithm design, *Sci. Computer Programming* **26** (1996) 15–31.

[5] J.-Y. Girard, Linear logic, *Theoretical Computer Sci.* **50** (1987) 1–101.

[6] J.-Y. Girard, Normal functors, power series and λ-calculus, *Ann. Pure Applied Logic* **37** (1988) 129–177.

[7] C. A. Gunter, *Semantics of Programming Languages. Structures and Techniques*, Foundations of Computing Series, (MIT Press, 1992).

[8] R. Hasegawa, Categorical data types in parametric polymorphism, *Mathematical Structures in Computer Science* **4** (1994) 71–109.

[9] R. Hasegawa, Relational limits in general polymorphism, *Publications of Research Institute for Mathematical Sciences* **30** (1994) 535–576.

[10] R. Hasegawa, A logical aspect of parametric polymorphism, in: *Computer Science Logic*, 9th International Workshop CSL'95, H. K. Büning, ed., Paderborn, Germany, 1995, Lecture Notes in Computer Science 1092, (Springer, 1995) pp. 291–307.

[11] Z. Hu, H. Iwasaki, and M. Takeichi, Deriving structural hylomorphisms from recursive definitions, in: *ACM SIGPLAN International Conference on Functional Programming*, Philadelphia, USA, (ACM Press, 1996) pp. 73–82.

[12] A. Joyal, Une théorie combinatoire des séries formelles, *Advances Math.* 42 (1981) 1–82.

[13] A. Joyal, Foncteurs analytiques et espèces de structures, *Combinatoire Enumérative*, Proceedings, Montreal, Québec, Canada, 1985, G. Labelle, P. Leroux, eds., Lecture Notes in Mathematics 1234, (Springer, 1986) pp. 126–159.

[14] G. M. Kelly and R. Street, Review of the elements of 2-categories, in: *Category Seminar*, Sydney, Australia, 1972/1973, Lecture Notes in Math. 420, (Springer, 1974) pp. 75–103.

[15] J. C. Mitchell, *Foundations for Programming Languages*, Foundations of Computing Series, (MIT Press, 1996).

[16] G. Plotkin, Second order type theory and recursion, unpublished notes, 1993.

[17] D. Rémy and J. Vouillon, Objective ML: An effective object-oriented extension to ML, *Theory Practice Object Systems* 4 (1998) 27–50.

[18] D. S. Scott, Continuous Lattices, in: *Toposes, Algebraic Geometry and Logic*, Halifax, Canada, 1971, Lecture Notes in Mathematics 274, (Springer, 1972) pp. 97–136.

[19] M. Suzuki, *Group Theory, I*, Grundlagen der mathematischen Wissenshaften 247, (Springer, 1982).